University of Puerto Rico
Mayaguez Campus
Department of Electrical and Computer Engineer

ChromeIt
Final Report

Cortés Cortés, Jaime A.
Ortiz Sacarello, Fernando J.
Perez Pagan, Luis R.

ICOM4036/CIIC4030
Prof. Wilson Rivera Gallego

Introduction

About Razer Chroma

Razer is a company based in San Diego, California that has the mission to create the best products for the gaming industry. One of the most prestigious products from Razer are their peripherals. Ranging from headphones, keyboards and mice, they have a huge product base which many believe are the top of the line of its category. One of its most reputable line of products is their Razer ChromaTM brand which consists of peripherals with fully customizable RGB LED lighting. Razer has various methods of customizing your device's lights including a GUI profile creator and an SDK to integrate the Chroma features with your own software.

Why should I ChromeIt!?

Razer Chroma SDK is developed for C/C++/C# which for some programmers and people with limited programming knowledge can be extremely tedious. The goal of creating a programming language like ChromeIt is making the ChromaTM integration very easy for individuals who are not familiar with the previously mentioned languages. With easy syntax you can still create profiles using code rather than the GUI, without the need to struggle learning a complex language like C/C++/C#.

About the Language

Language Features

The purpose of the ChromeIt language is to bring simplicity to its users in order to achieve the results they want within the Chroma ecosystem. This is achieved with the inclusion of some key features to shorten the bridge between implementation and execution. Such language features include:

- Translate simple grammar to complex C++/C/C# code to work with Chroma SDK
- Create a custom chroma animation profile
- Profiles usable for Razer Blackwidow Chroma and Razer Deathadder Chroma

Language Tutorial

The code consists of the following:

1. Creating effects
2. Playing the effects

Therefore, the code will be divided in these two parts:

Creating effects

Creating effects would be the hardest part of the language, but don't get alarmed because once again, it's simple.

Creating an effect consists of a line with various elements:

<EffectID> <EffectDevice> <EffectType> <EffectArguments>

Note: To learn what are the ***valid EffectID, EffectDevice, EffectType and EffectArgument, please consult the Reference Manual.***

Once these are written you can jump to the next line and create the next effect. You can create as many as you want.

Therefore, your effects creation part will have the following structure:

<EffectID> <EffectDevice> <EffectType> <EffectArguments>
<EffectID> <EffectDevice> <EffectType> <EffectArguments>
<EffectID> <EffectDevice> <EffectType> <EffectArguments>
<EffectID> <EffectDevice> <EffectType> <EffectArguments>

Or with actual valid code:

e2 MOUSE BLINK 200 200 RED BLUE
e3 MOUSE SPECTRUM 1
e4 MOUSE CUSTOM TOP RED BOTTOM BLUE
e5 KEYBOARD STATIC RED

Effects Creation Break:

To divide the creation and the play section, ChromeIt! uses double colon “::”
So in the next line after effect creations insert “::” just like:

<EffectID> <EffectDevice> <EffectType> <EffectArguments>
<EffectID> <EffectDevice> <EffectType> <EffectArguments>
<EffectID> <EffectDevice> <EffectType> <EffectArguments>
<EffectID> <EffectDevice> <EffectType> <EffectArguments>
::

Or

e2 MOUSE BLINK 200 200 RED BLUE
e3 MOUSE SPECTRUM 1
e4 MOUSE CUSTOM TOP RED BOTTOM BLUE
e5 KEYBOARD STATIC RED
::

Effects Playlist

In the line after the double colon you can start writing down your playlist.

Each effect play command consists of the following structure:

<EffectID> <EffectDuration>

In this section you can call all the effect you have created. The effects can be called none, once, or multiple times.

Therefore your effects playlist part will have the following structure:

<EffectID> <EffectDuration>
<EffectID> <EffectDuration>
<EffectID> <EffectDuration>
<EffectID> <EffectDuration>
Or with actual code

e2 5000
e3 5000
e4 5000
e5 5000

Real Code Example

```
e2 MOUSE BLINK 200 200 RED BLUE
e3 MOUSE SPECTRUM 1
e4 MOUSE CUSTOM TOP RED BOTTOM BLUE
e5 KEYBOARD STATIC RED
::
e2 5000
e3 5000
e4 5000
e5 5000
```

The code written above:

Creates the following effects:

1. Mouse effect with ID e2, type BLINK, ON time 200ms, OFF time 200ms, with RED and BLUE colors.
2. Mouse effect with ID e3, type SPECTRUM, time between colors 1ms.
3. Mouse effect with ID e4, type CUSTOM, TOP zone color RED, BOTTOM zone color BLUE.
4. Keyboard effect with ID e5, type STATIC, color RED.

With the following playlist:

1. Effect e2 for 5000ms
2. Effect e3 for 5000ms
3. Effect e4 for 5000ms
4. Effect e5 for 5000ms

Language Reference Manual

Applications Included:

A) ChromeItApplication.py - This version of ChromeIt includes the translator for ChromeIt Code (lexer, parser and intermediate code); it generates a folder with the C++ files ready for compilation. It DOES NOT generate the RazerChromaApplication.exe. THIS IS THE VERSION WHICH FULFILLS ICOM4036 PROJECT REQUIREMENTS.

B) ChromeItApplicationWithCComplier.py - This version of ChromeIt Includes everything in version A but additionally attempts to compile the code and create an executable with the RazerChromaApplication

Requirements

Version A:

-Python 2.7 must be installed

-ChromeIt Code written in 'ChromeItCode.txt'

-Translated code will be ready to compile in ChromeItCompilable

-Razer ChromaSDK must be installed prior to C/C++ compilation

Additional Requirements and notes for Version B:

-Project must be located in C: drive

-Visual Stduio C++ Enterprise or superior version with MFC support Must be Installed

-Razer Chroma SDK must be installed

-Path to folder "tools" of visual studio must be in the system's environment variables

-Output .exe will be stored in ChromeItCompilable folder

Application Execution

After completing the requirements mentioned above run the desired version:

ChromeItApplication.py or ChromeItApplicationWithCComplier.py

Code Rules

EffectID

Effect IDs consist of a string consisting of a lower-case letter followed by any letter of number

Valid examples:

iD1234
id1234
`i1D2i3o4'

not Valid:

ID1234
1234ID

EffectDevice

EffectDevice defines the device the effect will be created for.

Valid Tokens:

MOUSE - Use this to create effect for Chroma enable mice.
KEYBOARD - Use this to create effect for Chroma enable keyboards.

EffectType

EffectType defines the type of the effect that will be created.

Valid Tokens for MOUSE and KEYBOARD devices:
STATIC - All the lights on the device light up.

BLINK - All the lights on the device light on and off.

SPECTRUM - All the lights on the device light up through all the colors in the spectrum.

CUSTOM - The selected keys or zones lights up on the desired color.

Valid Tokens for KEYBOARD devices ONLY:

WAVE - All the lights on the device light up forming a wave.

BREATHE - All the lights on the device slowly light on and off on the desired color.

REACT - Keys react to presses on the desired color.

STARLIGHT - Random keys light on and off every 175ms.

EffectArguments

Every EffectType has its unique set of EffectArguments. All Arguments must be specified in the right order

STATIC Arguments:

<COLOR>

BLINK Arguments:

<TimeON> <TimeOFF> <COLOR> <COLOR>

SPECTRUM Arguments:

<TimeBetweenColors>

CUSTOM Arguments:

Repetition of: <Key or Zone> <COLOR>

WAVE Arguments:

<DIRECTION>

BREATHE Arguments:

<BreatheMode> <COLOR> *Check EffectModes Section for details.

REACT Arguments:

<ReactSpeed> <COLOR>

STARLIGHT Arguments:

<KeysCount> <COLOR>

COLORS

The Following colors are valid tokens:

RED, BLACK, WHITE, GREEN, BLUE, YELLOW, PURPLE, CYAN, ORANGE, PINK, GREY

Or it can be specified in RGB format:

RGB(###,###,###)

Example: RGB(255, 255, 255)

NOTE:

When <COLORS> is used, you can choose to use one or more colors e.g. RED BLUE GREEN

TIME

Arguments such as TimeON, TimeOFF and TimeBetweenColors are specified in Miliseconds

TimeON - Specifies how long the LEDs will be on during the BLINK effect

TimeOFF - Specifies how long the LEDs will be off during the BLINK effect

TimeBetweenColors - Specifies how long it takes for the LEDs to change color during the SPECTRUM effect

DIRECTION

This Argument defines the direction in which the WAVE effect will move

L2R - Wave will move from left to right

R2L - Wave will move from right to left

Effect Modes

<ReactSpeed> - Defines how long the light will stay on

<BreatheMode> - Defines the type of Breath the device will use

####<ReactSpeed> valid Values:

SHORT - lights stay on for a short time

MEDIUM - lights stay on for a medium amount of time

LONG - lights stay on for a long time

<BreatheMode>

TWOCOLORS - The effect will breathe between two specified colors

RANDOM - the effect will breathe between random colors. *For this option, no colors will be specified

Keys and Zones

KeysCount - The number of keys that will light up during the STARLIGHT effect

Zones

Refers to the zones of the MOUSE Device

Valid Tokens:

TOP, MIDDLE, BOTTOM

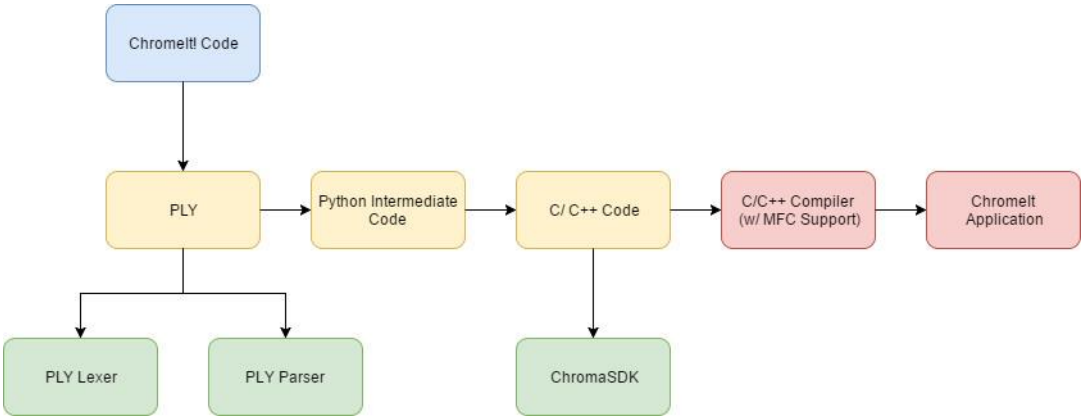
Keys

RZKEY_NUMLOCK - NUMLOCK Key
RZKEY_NUMPAD - NUMPAD #0
RZKEY_NUMPAD2 - NUMPAD #2
RZKEY_NUMPAD3 - NUMPAD #3
RZKEY_NUMPAD4 - NUMPAD #4
RZKEY_NUMPAD5 - NUMPAD #5
RZKEY_NUMPAD6 - NUMPAD #6
RZKEY_NUMPAD7 - NUMPAD #7
RZKEY_NUMPAD8 - NUMPAD #8
RZKEY_NUMPAD9 - NUMPAD #9
RZKEY_NUMPAD_DIVIDE - NUMPAD /
RZKEY_NUMPAD_MULTIPLY - NUMPAD *
RZKEY_NUMLOCK - NUMLOCK Key
RZKEY_NUMPAD_SUBTRACT - NUMPAD -
RZKEY_NUMPAD_ADD - NUMPAD +
RZKEY_NUMPAD_ENTER - NUMPAD ENTER
RZKEY_NUMPAD_DECIMAL - NUMPAD .
RZKEY_PRINTSCREEN - PRINTSCREEN Key
RZKEY_SCROLL - SCROLL Key
RZKEY_PAUSE - PAUSE Key
RZKEY_INSERT - INSERT Key
RZKEY_HOME - HOME Key
RZKEY_PAGEUP - PAGEUP Key
RZKEY_DELETE - DELETE Key
RZKEY_END - END Key
RZKEY_PAGEDOWN - PAGEDOWN Key
RZKEY_UP - UP Key
RZKEY_LEFT - LEFT Key
RZKEY_DOWN - DOWN Key
RZKEY_RIGHT - RIGHT Key
RZKEY_TAB - TAB Key
RZKEY_CAPSLOCK - CAPSLOCK Key
RZKEY_BACKSPACE - BACKSPACE Key
RZKEY_ENTER - ENTER Key
RZKEY_LCTRL - LEFT CONTROL Key
RZKEY_LWIN - LEFT WINDOWS Key
RZKEY_LALT - LEFT ALT Key
RZKEY_SPACE - SPACE Key
RZKEY_RALT - RIGHT ALT Key
RZKEY_FN - FN Key
RZKEY_RMENU - RIGHT MENU Key
RZKEY_RCTRL - RIGHTR CONTROL Key
RZKEY_LSHIFT - LEFT SHIFT Key
RZKEY_RSHIFT - RIGHT SHIFTKey
RZKEY_MACRO1 - MACRO1 Key
RZKEY_MACRO2 - MACRO2 Key
RZKEY_MACRO3 - MACRO3 Key
RZKEY_MACRO4 - MACRO4 Key
RZKEY_MACRO5 - MACRO5 Key
RZKEY_OEM_1 - OEM1 Key
RZKEY_OEM_2 - OEM2 Key
RZKEY_OEM_3 - OEM3 Key
RZKEY_OEM_4 - OEM4 Key
RZKEY_OEM_5 - OEM5 Key
RZKEY_OEM_6 - OEM6 Key
RZKEY_OEM_7 - OEM7 Key
RZKEY_OEM_8 - OEM8 Key
RZKEY_OEM_9 - OEM9 Key
RZKEY_OEM_10 - OEM10 Key
RZKEY_OEM_11 - OEM11 Key
RZKEY_EUR_1 - EUR1 Key
RZKEY_EUR_2 - EUR2 Key
RZKEY_JPN_1 - JPN1 Key
RZKEY_JPN_2 - JPN2 Key
RZKEY_JPN_3 - JPN3 Key
RZKEY_JPN_4 - JPN4 Key
RZKEY_JPN_5 - JPN5 Key
RZKEY_KOR_1 - KOR1 Key
RZKEY_KOR_2 - KOR2 Key
RZKEY_KOR_3 - KOR3 Key
RZKEY_KOR_4 - KOR4 Key
RZKEY_KOR_5 - KOR5 Key
RZKEY_KOR_6 - KOR6 Key
RZKEY_KOR_7 - KOR7 Key
RZKEY_ESC - ESC Key
RZKEY_F1 - F1 Key
RZKEY_F2 - F2 Key
RZKEY_F3 - F3 Key
RZKEY_F4 - F4 Key
RZKEY_F5 - F5 Key
RZKEY_F6 - F6 Key
RZKEY_F7 - F7 Key
RZKEY_F8 - F8 Key
RZKEY_F9 - F9 Key

RZKEY_F10 - F10 Key
RZKEY_F11 - F11 Key
RZKEY_F12 - F12 Key
RZKEY_1 - 1 Key
RZKEY_2 - 2 Key
RZKEY_3 - 3 Key
RZKEY_4 - 4 Key
RZKEY_5 - 5 Key
RZKEY_6 - 6 Key
RZKEY_7 - 7 Key
RZKEY_8 - 8 Key
RZKEY_9 - 9 Key
RZKEY_0 - 0 Key
RZKEY_A - A Key
RZKEY_B - B Key
RZKEY_C - C Key
RZKEY_D - D Key
RZKEY_E - E Key
RZKEY_F -F Key
RZKEY_G - G Key
RZKEY_H- H Key
RZKEY_I - I Key
RZKEY_J - J Key
RZKEY_K - K Key
RZKEY_L - L Key
RZKEY_M - M Key
RZKEY_N - N Key
RZKEY_O - O Key
RZKEY_P- P Key
RZKEY_Q - Q Key
RZKEY_R - R Key
RZKEY_S - S Key
RZKEY_T - T Key
RZKEY_U - U Key
RZKEY_V - V Key
RZKEY_W- W Key
RZKEY_X - X Key
RZKEY_Y - Y Key
RZKEY_Z - Z Key

Language Development

Translator Architecture



ChromeIt Code

This is the code written by the user. Here the user specifies the desired effects and playlist.

PLY

PLY is an implementation of lexer and parser tools for python.

Lexer

The Lexer tokenizes everything the user wrote in the code and sends it to the parser.

Parser

The parser verifies the language grammar rules and sends the desired requests to the python intermediate code.

Python Intermediate Code

This code receives the requests from the parser and proceeds to generate the C/C++ to be written as C/C++ source .

C/C++ Code

This is the heart of the ChromeIt Application. Here the calls for the effects are received and generated using the ChromaSDK which enables playing the effects on the Chroma enabled devices.

C/C++ Compiler and ChromeIt Application

Although not part of the translator perse, the compiler takes the C/C++ code and generates the ChromeIt Application which can be executed to play the effects on the Chroma enabled devices.

Interfaces Between Modules

The ChromeIt! Architecture works the following way:

1. First the User writes code in the ChromeIt Language.
2. Following, when the user runs the ChromeIt translator which reads the ChromeIt Code and is sent to the PLY Lexer whcih tokenizes the code and then to the PLY parser which parses it.

3. After all data has been obtained from the code, it is sent to the Python intermediate code which categorizes and translates it into C/C++ source code.
4. The C/C++ code creates the effects and playlists making use of the C/C++ intermediate code and Razer's ChromaSDK.
5. After this code is ready, it is ready to be compiled using a C/C++ compiler with MFC support (Visual C++ recommended)
6. The generated ChromeIt Application can be executed to play the effects on Chroma Enabled devices

Software Development Environment

One interesting aspect of ChromeIt is that it mixes languages, including C/C++ and Python. For the development of the parts in each language, a different IDE was used.

PyCharm

Used for the development of the part of the ChromeIt translator in python.

Visual Studio 2017

Used for the development intermediate code in C/C++.

Test Methodology

Lexer

The first thing we developed was the lexer. For this part, most of the testing was done by feeding the lexer with desired tokens and verifying that our tool recognized these tokens properly.

Parser

After defining our grammar rules, we developed our parser. For each part of this tool we developed, we sent it our desired line of ChromeIt code and checked for errors.

Python to C/C++

This part wasn't too tedious, it consisted of writing Strings into a text file, so if there weren't any typos, there was no errors at all. If any errors were made, the C/C++ compiler would let us know.

C/C++ Intermediate Code

This code was tested by compiling the C/C++ code and running it on the Chroma enabled devices to confirm the effects were running correctly.

Programs Used to Test Translator

Most testing was done as described in the above section, but halfway through the development we developed a python script **ChromeItApplicationWithCCompiler.py** to automate the translating and C/C++ compiling so that we could test the effect more efficiently. This script generated the Effects Application which we could easily run and verify the effects were playing correctly on the Chroma Devices.

Conclusion

Developing a new programming language is a great experience to understand how these work. Such task is a very complex process which starts with an idea which one has to understand, expand, find a solution and finally, implement it. From the beginning, the main reason for developing ChromeIt, was to free programmers that doesn't feel comfortable with the C/C++ Language, but being one of those programmers ourselves, we had a steep learning curve we had to surpass. Even better, while it may sound odd, it was an incredible experience mixing various programming languages to fulfill our goals. By selecting a lexer/parser in a language different than our target we dove into even deeper water. Nonetheless, mixing these many elements in the creation of our new programming language, made it possible for us to expand our knowledge in the field of computer sciences while helping others get through barriers.